Physically Unclonable Pseudorandom Functions

Frederik Armknecht†, Roel Maes‡, Ahmad-Reza Sadeghi†, Berk Sunar∗, Pim Tuyls‡

† Horst Görtz Institute for IT Security, Ruhr-University Bochum, Germany
‡ ESAT/COSIC, Catholic University of Leuven, Belgium
∗ Cryptography & Information Security, WPI, MA USA

Abstract. With the proliferation of physical attacks the implicit assumptions made in traditional security models no longer reflect the real world. To address this issue, a number of new security models, e.g. Algorithmic Tamper-Proof Security, have been proposed. In this work, we take another step and identify the cryptographic properties of a particular family of physical functions, termed as Physically Unclonable Functions (PUFs), that exploit physical phenomena at deep-submicron and nano-scale level. PUFs provide low-cost tamper-evident and tamper-resilient implementations. Motivated by this fact, we specifically describe a general method for constructing Pseudorandom Functions (PRFs) from a class of PUFs. We provide a formal model for certain types of PUFs that build the basis for PRFs, which we call PUF-PRFs. Furthermore, we show experimentally that some real world PUF instantiations (e.g., SRAM PUFs) satisfy the model. This strongly indicates that PUF-PRFs can indeed be physically realized.

Keywords: Physically Unclonable Functions, Pseudorandom Functions

1 Introduction

Modern cryptography provides us with a variety of useful tools and methodologies to analyze and to prove the security of cryptographic schemes within certain attack models. Prominent approaches are game-based security framework, e.g., see [3–6], and Universal Composibility (UC) framework [7] to name some.

These theoretical frameworks focus mainly on the algorithmic aspects under the so-called black-box assumption. Here the dishonest parties are under the control of the adversary whereas for the honest parties it is assumed that their sensitive information, e.g., key material, can be kept secret. However, real world cryptographic applications are embedded in physical environments that may leak by physical signals sensitive information about the performed computation, in

One may debate whether the notion of “provable security” is the ultimate approach for practice (see, e.g., [1, 2]). However, we believe that provable security approaches strongly improved the understanding of cryptography and the underlying assumptions.
particular about the cryptographic key material processed. Hence even honest parties may leak some security critical information. A large body of literature has therefore explored various forms of side-channel attacks and analysis (e.g., see [8–10]). This implies that new security models are required to close the gap between the black-box approach and real-life, and it motivates new security definitions that try to capture the effect of the (physical) information leakage. This aspect has been considered in the literature to some extent, e.g., by the concept of Physically Observable Cryptography (see, e.g.,[11]), however, again, under some (unavoidable) assumptions such as more or less idealized physical leakage channel.

Hence, when talking about the security of real world applications we are practically concerned with some physical assumptions regarding secure storage of the underlying key material by assuming tamper-resistance, tamper-proofness or tamper-evidence storage (black-box). This is certainly in addition to the hardness assumptions (e.g., for public key schemes) or the assumption that some heuristic design principles are essential for security aspects (e.g., in case of many symmetric schemes).

Physical assumptions have been used to relax some impossibility results that hold under computational assumptions within the black-box model. For instance it is known that without any further assumptions secure general multiparty computation is impossible if the majority of the involved parties is dishonest. Hence, the existing solutions commonly make different setup assumptions that in practice require some trusted party for the system initialization. A recent exciting development to relax this requires a system setup based on the following physical assumption: the existence of tamper proof hardware [12–14].

Another recent strand of research aims at developing methods to protect against compromise of key material at the physical level, more precisely deep-submicron and nano-scale physical phenomena are used to build low-cost tamper-evident and tamper-resistant key storage devices [15–17]. One of the most promising approaches in this context are the primitives called Physical Unclonable Functions (PUFs) introduced in [15, 18] and further developed, e.g., in [19]. A PUF is a primitive that maps challenges to responses which are highly dependent on the physical properties of the device in which the PUF is contained or embedded.

Basically a PUF has a physical part that consists of many random components that are uncontrollable even by the manufacturer. The random structure stems from uncontrollable process variations during the manufacturing [20–23]. For some PUF instantiations it is assumed to be infeasible to copy (clone) these

---

2 Note that key material can also be protected by means of obfuscation mechanisms (white-box). However, the general belief is that hardware protection is stronger than software protection.

3 Certainly, one may debate on how realistic the strong assumption of tamper-proofness in practice is, and moreover, whether in real life applications one can avoid trust in some parties such as hardware and platform manufacturers.

2
random structures. In the case of PUFs on an Integrated Circuit (IC) such process variations are typically deep-submicron variations such as doping variations in transistors. Based on this property each PUF has its own unique functional behavior containing a high amount of entropy. Examples of PUFs include optical PUFs [15, 18], silicon PUFs [24], coating PUFs [16], Intrinsic-PUFs [25], and LC-PUFs [26].

The unclonability property of some PUF instantiations have made PUFs very appealing for IP protection and secure key storage applications [16, 27, 28, 25]: more concretely (i) practically no key material needs to be stored in non-volatile memory since keys can be re-generated at any time they are needed, (ii) keys are bound to the (physical properties of) underlying hardware, (iii) the security and adversary model is extended beyond the black-box model and based on physical assumptions at nano-scale level.

Meanwhile PUFs have been proposed for constructing various cryptographic primitive. For instance, the metastability phenomena in Arbiter PUFs was exploited to construct a True Random Number Generator (TRNG) [29, 30] or authentication schemes [30, 31].

In this paper, we explore the construction of pseudorandom functions from PUFs. Pseudorandom functions (PRFs), e.g., see [32], are a crucial concept in cryptography and have many applications, e.g., see [33–35]. The existence of PRFs can be shown to be equivalent to the existence of other cryptographic primitives like one-way functions or pseudorandom generators (for an overview, see [36]) and methods are known on how to construct one from the other. However, for practical reasons, PRFs are often either constructed from scratch or from stronger (mostly number theoretic) assumptions like the Decisional Diffie-Hellman assumption. Observe that, to the best of our knowledge, all construction methods for PRFs so far are purely software-based. That is, no hardware-based PRFs have been proposed yet.

Our contribution. In this paper, we describe a general method for constructing PRFs from PUFs that meet well defined properties. We provide a formal model for such PUFs and explain how PRFs can be built from them, which we call PUF-PRFs. Furthermore, we show experimental results that some real world instantiations of PUF (e.g., SRAM PUFs [37]) have the required properties. Hence, our construction has both theoretical and practical value.

The proposed constructions are provably secure under reasonable physical assumptions. The unclonability property of some PUFs is beneficial from different perspectives: on the one hand it makes PUFs perfectly suitable for specific applications such as personal data encryption like the hard disk of a PC. One could imagine of having the encryption scheme implemented in an external device.\(^4\)

\(^4\) Note that this stands in sharp contrast to Quantum Cryptography where cloning is impossible due to the basic laws of nature. In the case of PUFs based on classical physics, there is a very small probability that the structure can be cloned, it is however estimated to take a long time and carry a lot of costs.

\(^5\) or being part of a Trusted Platform Module (TPM) [17] in a PC.
Then, only those parties having access to the PUF can actually decrypt. On the other hand the construction naturally provides protection against hardware attacks. Hence, no additional protection mechanisms are necessary. Furthermore, since no keys are needed, the user is released from the burden of memorizing any secret. In particular, this might help to improve protection against social attacks like phishing.

A related topic in this context is True Random Number Generators (TRNGs) which also exploits physical phenomena to generate randomness. However, as opposed to our construction, TRNGs cannot be used as functions: challenging a TRNG twice with the same input does not yield the same (or at least similar) outputs. This disqualifies TRNGs for the considered cryptographic applications.

Overview of our construction

The unclonability property of some PUF instantiations already guarantees that PUF are good sources for randomness. Furthermore, as opposed to TRNGs, PUFs are functions and do always respond to the a challenge with the (almost) same answer. However, PUFs differ in two aspects significantly from PRFs. First, their responses are noisy by nature, i.e., two calls to the PUF with the same challenge will produce two different but closely related responses, where the measure of closeness can be defined via a distance function. Secondly, the outputs are not necessarily uniformly distributed. In order to derive reliable and uniform strings from PUFs, we make use of the concept of Fuzzy Extractors (e.g., see [38, 39]).

Organization. This paper is organized as follows. In Section 2 we provide some preliminaries. In Section 3, we give a formal model for (certain types of PUFs) and show how these can be turned into pseudorandom functions. After that, we describe one example instantiation of PUFs, namely SRAM PUFs, and argue why these fit to the model. Finally in Section 6 we present the conclusions.

2 Preliminaries

Distributions: For a probability distribution \( \mathbb{D}_m \) on \( \{0,1\}^m \), the expression \( x \leftarrow \mathbb{D}_m \) denotes the event that \( x \) has been sampled according to \( \mathbb{D}_m \). If \( m \) is clear from the context, we simply write \( \mathbb{D} \) and \( \mathcal{O}^{\mathbb{D}} \). For \( m \geq 1 \), we denote by \( \mathbb{U}_m \) the uniform distribution on \( \{0,1\}^m \). The min-entropy \( H_{\infty}(\mathbb{D}) \) of a distribution \( \mathbb{D} \) is defined by \( H_{\infty}(\mathbb{D}) := -\log_2(\max_x Pr[x \leftarrow \mathbb{D}]) \). The statistical distance between two distributions \( \mathbb{D}_m \) and \( \mathbb{D}_m' \) on \( \{0,1\}^m \) is the value

\[
\text{SD}(\mathbb{D}, \mathbb{D}') \triangleq \frac{1}{2} \sum_x |Pr[x \leftarrow \mathbb{D}] - Pr[x \leftarrow \mathbb{D}']|.
\]  

Boolean functions and oracles: For positive integers \( \ell, m \in \mathbb{N} \), let \( F_{\ell,m} \) denote the set of all Boolean functions from \( \{0,1\}^{\ell} \) to \( \{0,1\}^m \). If \( \ell = m \), we abbreviate \( F_{\ell,\ell} \) to \( F_{\ell} \). For any subset \( \mathcal{F} \subseteq F_{\ell,m} \), the oracle \( \mathcal{O}^\mathcal{F} \) works as follows. First, it uniformly random picks a function \( f \in \mathcal{F} \). After that, it accepts inputs

\[6\] The downside is, however, that PUF-based symmetric schemes cannot be used to encrypt data for a distant recipient.
\( x \in \{0,1\}^f \) and responds with \( f(x) \). For \( f \in \mathbb{F}_{\ell,m} \) and \( g \in \mathbb{F}_{m,n}, g \circ f \in \mathbb{F}_{\ell,n} \) denotes their composition, that is \((g \circ f)(x) \stackrel{\text{def}}{=} g(f(x))\).

**Distinguisher:** Let \( \mathcal{O} \) and \( \mathcal{O}' \) denote two oracles with the same input and output domain. A *distinguisher* \( D = D(\mathcal{O}, \mathcal{O}', q) \) is a (possibly probabilistic) algorithm that has either access to the oracle \( \mathcal{O} \), denoted by \( D_\mathcal{O} \), or to the oracle \( \mathcal{O}' \), denoted by \( D_{\mathcal{O}'} \). The distinguisher \( D \) can make up to \( q \) queries to the associated oracle and has finally to decide if it was communicating with \( \mathcal{O} \) or \( \mathcal{O}' \). The advantage of \( D \) is defined by

\[
\text{Adv}(D) \stackrel{\text{def}}{=} |\Pr[1 \leftarrow D_\mathcal{O}] - \Pr[1 \leftarrow D_{\mathcal{O}'}]|.
\]

Furthermore, we define \( \text{Adv}(\mathcal{O}, \mathcal{O}', q) \stackrel{\text{def}}{=} \max_D \text{Adv}(D) \).

The notion of pseudorandom functions (PRFs) and pseudorandom permutations (PRP) is established since long in cryptography and many important cryptographic schemes can be constructed from them. As we aim in this paper for concrete realizations, we consider here a definition for finite PRFs and PRPs. The definition is based on the indistinguishability of certain oracles:

**Definition 1 (Pseudorandom functions and permutations).** For \( \ell, n \in \mathbb{N}^* \), we define an oracle \( \mathcal{O}_{PRF}^{\ell,n} \) as follows. \( \mathcal{O}_{PRF}^{\ell,n} \) accepts input-queries \( x \in \{0,1\}^\ell \) and checks if \( x \) has been queried before. If not, it returns a random value \( y \leftarrow \mathbb{U}_n \). Otherwise, it repeats the same response which it gave the first time when \( x \) was queried.

A family of functions \( \mathcal{F} \subseteq \mathbb{F}_{\ell,n} \) is called a family of \((\ell, n, q, \epsilon)\)-pseudorandom functions if \( \text{Adv}(\mathcal{O}_\mathcal{F}, \mathcal{O}_{PRF}^{\ell,n}, q) \leq \epsilon \). A family of permutations \( \mathcal{P} \subseteq \mathbb{F}_n \) is called a family of \((n, q, \epsilon)\)-pseudorandom permutations if \( \text{Adv}(\mathcal{O}_\mathcal{P}, \mathcal{O}_{PRP}^{n,n}, q) \leq \epsilon \).

The oracle \( \mathcal{O}_{PRP}^n \) accepts two types of queries: input-queries \( x \) and output-queries \( y \), both coming from the domain \( \{0,1\}^n \). It keeps record of all occurred input/output pairs in a set \( Q \) as follows. Initially, \( Q \) is empty. On an input-request \( x \) it first checks if \( (x,y) \in Q \) for some \( y \). In the positive case, it responds with \( y \). Otherwise, it chooses a value \( y \leftarrow \mathbb{U}_n \), stores \( (x,y) \) in \( Q \) and outputs \( y \). In a similar manner, for every output-query \( y \), it is first verified whether \( (x,y) \in Q \) for some \( x \). If this is the case, \( x \) is given out. Otherwise, \( x \leftarrow \mathbb{U}_n \) is sampled, the pair \( (x,y) \) stored in \( Q \), and \( x \) given as a response. A family of permutations \( \mathcal{P} \subseteq \mathbb{F}_n \) is called a family of \((n, q, \epsilon)\)-strong pseudorandom permutations if \( \text{Adv}(\mathcal{O}_\mathcal{P}, \mathcal{O}_{PRP}^n, q) \leq \epsilon \).

We call a function a PRF, PRP, or strong PRP if it is a member of a family of PRFs, PRPs, or strong PRPs, respectively.

Observe that this definition differs slightly from the definition of pseudorandom functions given in [40, 41]. There, oracles were considered where a function \( f \) is initially chosen at random and then each queries are responded using \( f \). Here, the oracles randomly choose the responses itself. It is an easy task to show that both definitions are equivalent.
3 Pseudorandom functions build from PUFs

One contribution of this work at hand is to construct PRFs from physical mechanisms called physically unclonable functions (PUFs) which map some challenges in \( \{0,1\}^\ell \) to responses in \( \{0,1\}^m \). The main properties of PUFs important in this context are:

1. PUFs are probabilistic (or noisy) in that sense that the same challenges can lead to different responses.
2. The outputs of PUFs are not necessarily uniformly distributed which is mandatory for a PRF.

Before we explain which mechanisms are helpful for the realization of PRFs based on PUFs, we provide next a formal model for PUFs. We start with definition for noisy functions:

**Definition 2 (Noisy functions).** For two positive integers \( \ell, m \in \mathbb{N} \) and a value \( \delta \in \{0, \ldots, m\} \), a \((\ell, m, \delta)\)-noisy function \( f^* \) is a probabilistic algorithm with the following properties:

1. \( f^* \) accepts inputs (challenges) from \( \{0,1\}^\ell \) and outputs (responses) from \( \{0,1\}^m \).
2. There exists a Boolean function \( f \in F_{\ell,m} \) such that for each query \( x \in \{0,1\}^\ell \), the response \( y \in \{0,1\}^m \) has the form \( y = f(x) \oplus e \) where \( e \) is some random noise vector. In other words, the responses can be interpreted as being the outputs of \( f \) transmitted via an additively noisy channel. The vector \( e \) is called the noise vector.
3. All noise vectors have a Hamming weight of \( \delta \) or less can occur.

In a similar manner, we define \((\ell, m, \delta)\)-noisy family of functions to be a set of \((\ell, m, \delta)\)-noisy functions.

For the second property, the non-uniform output, the min-entropy of the output distribution tells how many random bits can be extracted from the outputs. In our context we are interested into the min-entropy of the output distributions over several queries which might be under the control of an adversary. This will be addressed in our model:

**Definition 3 (Physically Unclonable Functions).** A \((\ell, m, \delta, q, \mu)\)-family of PUFs is a \((\ell, m, \delta)\)-noisy family of functions \( F \) with the following additional property. There exists a distribution \( \mathbb{D} \) on \( \{0,1\}^m \) with \( \text{H}_\infty(\mathbb{D}) \geq \mu \) such that it holds for all vectors \((x_1, \ldots, x_q) \in (\{0,1\}^\ell)^q \) with pairwise different elements that the following two distributions are identical:

1. \((f(x_1), \ldots, f(x_q)) \) where \( f \) is uniformly chosen from \( F \).
2. the distribution \( \mathbb{D}^q \) on \( \{0,1\}^m \), that is each entry is independently sampled according to \( \mathbb{D} \).

A \((\ell, m, \delta, q, \mu)\)-PUF is a member of a \((\ell, m, \delta, q, \mu)\)-family of PUFs.
We will argue in Section 4 why this model fits to at least certain classes of PUFs. That is, we show the justification for this model based on some reasonable physical assumptions.

Remark 1 (One-wayness). The second property implies that PUFs as defined in Definition 3 are one-way functions. Otherwise, if one could deduce from \( y = \Pi(x) \) some information on \( x \), the two distributions considered in Definition 3 would not be equal.

Observe that PUFs differ in two aspects from PRFs. Firstly, PUFs are noisy in that sense that querying the same challenge twice can lead to different (however close) results. Secondly, the output distribution over \( \{0,1\}^m \) might have a min-entropy that is smaller than \( m \) which means that the distribution differs from the uniform distribution. Hence, the task of construction a PRF from a PUF is related to the problem of converting noisy non-uniform inputs into reliably reproducible, uniformly random string. For this purpose, fuzzy extractors (FE) have been developed (see, e.g., [39]). First recall the definition of FE:

**Definition 4.** [39] A \((m, n, \delta, \mu, \epsilon)\)-fuzzy extractor \( E \) is a pair of randomized procedures, “generate” \((\text{Gen})\) and “reproduce” \((\text{Rep})\), with the following properties:

1. The generation procedure \( \text{Gen} \) on input \( y \in \{0,1\}^m \) outputs an extracted string \( z \in \{0,1\}^n \) and a helper string \( \omega \in \{0,1\}^* \).
2. The reproduction procedure \( \text{Rep} \) takes an element \( y' \in \{0,1\}^m \) and a bit string \( \omega \in \{0,1\}^* \) as inputs. The correctness property of fuzzy extractors guarantees that if the Hamming distance \( \text{dist}(y, y') \leq \delta \) and \( z, \omega \) were generated by \( (z, \omega) \leftarrow \text{Gen}(y) \), then \( \text{Rep}(y', \omega) = z \). If \( \text{dist}(y, y') > \delta \), then no guarantee is provided about the output of \( \text{Rep} \).
3. The security property guarantees that for any distribution \( D \) on \( \{0,1\}^m \) of min-entropy \( \mu \), the string \( z \) is nearly uniform even for those who observe \( \omega \): if \( (z, \omega) \leftarrow \text{Gen}(D) \), then it holds that \( \text{SD}((z, \omega), (U_n, \omega)) \leq \epsilon \).

We note that in the Def. of a FE the condition that \( H_\infty(D) \geq \mu \) can be replaced with the condition that the smooth min-entropy \( H_\epsilon^\infty(D) \) of \( D \) is \( \geq \mu \). This follows from the fact that the Left-Over Hash Lemma also holds when smooth min-entropy is used as was shown in [42]. For the definition of Smooth min-entropy we refer to [42].

In [39], several constructions for efficient fuzzy extractors have been presented. Given the existence of FE, the strategy is clear. To construct a PRF from PUFs, one first invokes the PUF and the applies an appropriate FE afterwards. This will be formalized in the following definition:

**Definition 5 (PUF-FE-composition).** Consider a \((\ell, m, \delta, q, \mu)\)-PUF \( \Pi \) and a \((m, n, \delta, \mu, \epsilon)\)-fuzzy extractor \( E = (\text{Gen}, \text{Rep}) \). We define their composition \( E \circ \Pi \) to be the following pair of randomized procedures:

---

7 We note that this modification is only important to motivate the experimental part and has no further impact on the theory.
**Generation:** The generation function $\text{Gen} \circ \Pi$ on input $x \in \{0,1\}^\ell$ outputs an extracted string $z \in \{0,1\}^n$ and a helper string $\omega_x \in \{0,1\}^*$. This is achieved by first querying the PUF $\Pi$ with $x$ to get a value $y \in \{0,1\}^m$ and then to apply the fuzzy extractor procedure $\text{Gen}$ on it.

**Reproduction:** The reproduction function $\text{Rep} \circ \Pi$ takes an element $x \in \{0,1\}^\ell$ and a bit string $\omega_x \in \{0,1\}^*$ as inputs and outputs a value $z' \in \{0,1\}^n$. At first, the PUF $\Pi$ is challenged with $x$ which gives a response $y' \in \{0,1\}^m$. Then, the final output $z' = \text{Rep}(y', \omega_x)$ is computed.

The following proposition shows the correctness of the reproduction procedure:

**Proposition 1 (Correctness).** Consider a PUF-FE composition $E \circ \Pi$ as specified in Definition 5. Then it holds for any challenge $x \in \{0,1\}^\ell$ and $(z, \omega_x) \overset{\text{def}}{=} (\text{Gen} \circ \Pi)(x)$ that $(\text{Rep} \circ \Pi)(x, \omega_x) = z$.

**Proof.** By assumption, querying the underlying PUF twice with the same challenge $x$ may lead to two different responses $y, y'$ but with a Hamming distance of at most $\delta$. Furthermore, the correctness property of fuzzy extractors guarantees that if $\text{dist}(y, y') \leq \delta$ and $z, \omega$ were generated by $(z, \omega) \leftarrow \text{Gen}(y)$, then $\text{Rep}(y', \omega) = z$. This shows that $(\text{Rep} \circ \Pi)(x, \omega_x) = z$. □

In the remainder of this section, we will discuss how PUF-FE-compositions can be used as a replacement for PRFs. This makes it necessary to define their mode of operation which we informally sketch now.

Consider a scheme $S^f$ which deploys a PRF $f$, that is a function that has been randomly chosen from a family of pseudorandom functions, and let $E \circ \Pi$ be a PUF-FE-composition with the same input and output domain as $f$. We describe now the scheme $S^{f/\Pi}$ where, in principle, each invocation of $f$ in $S^f$ has to be executed on an input $x$, the following procedure is started:

1. If $x$ has never been queried before, $(\text{Gen} \circ \Pi)$ is executed to produce a pair $(z, \omega_x)$. $z$ replaces the output of $f$ on $x$ in $S^f$ while the helper data $\omega_x$ is stored separately.\(^8\)
2. If $x$ has been queried before, then an according helper data $\omega_x$ has been recorded. In this case, $(\text{Rep} \circ \Pi)(x, \omega_x)$ is computed which, according to Proposition 1, yields the same value $z$ as in the first query.

Recall that a $\langle q, \epsilon \rangle$-PRF is characterized by the property, that an attacker who can make up to $q$ queries cannot decide with an advantage better than $\epsilon$ if he received the evaluation of the queries or some uniformly random values. In the following, we show that a similar property holds for PUF-FE-compositions. For this purpose, we introduce an adapted definition of the oracles which occur in definition 1. In principle, the oracles behave in a similar manner with the major difference being that each response contains two entries instead of one. The first entry is the one that is used as a substitute for the PRF output (see mode of operation sketched above) while the second entry is some helper data.

\(^8\) How the helper data is handled may depend on the considered scheme.
Definition 6. Let $\mathcal{F}$ be a family of PUF-FE-compositions. We define the following two oracles:

1. $O^F$ acts similarly as the oracle of the same name from section 2. That is at first it uniformly random picks a PUF-FE-composition $(E \circ \Pi) \in \mathcal{F}$. After that, it accepts inputs $x \in \{0, 1\}^\ell$ and gives out responses in $\{0, 1\}^n \times \{0, 1\}^*$. If $x$ is queried for the first time, the output is computed by $(z, \omega_x) = (\text{Gen} \circ \Pi)(x)$ and the output is stored. All subsequent queries of $x$ are then answered by the same response.

2. The oracle $O^{PRF, F}$ likewise chooses in a first step uniformly at random a PUF-FE-composition $(E \circ \Pi) \in \mathcal{F}$. For each (previously unasked) query $x$, a pair $(z, \omega_x) = (\text{Gen} \circ \Pi)(x)$ is computed. However, in contrast to $O^F$, then the value $z$ is replaced by a random value $z' \leftarrow \mathbb{U}_n$. The final output is $(z', \omega_x)$. If in subsequent queries the same value $x$ is asked again, then the same output is given back.

The following theorem now shows that given an appropriate family of PUF-FE-composition, an adversary cannot distinguish between the two oracles from Definition 6. In particular it holds that the first entries of the outputs from a PUF-FE-composition cannot be distinguished from uniformly random values, even if the associated helper data is known. As the first entry is the one that is used as a replacement for a PRF-output, this shows that the usage of PUF-FE-compositions as explained above actually gives an adequate replacement for "traditional" PRFs. We first recall the connection between the advantage of a distinguisher to distinguish two distributions $\mathbb{D}$ and $\mathbb{D}'$ and their statistical distance:

**Proposition 2.** Consider two distributions $\mathbb{D}$ and $\mathbb{D}'$ on the same domain $X$. Then it holds for any positive integers $\ell, q \in \mathbb{N}$ with $2^\ell \geq q$ that

$$\text{Adv}(O^{\mathbb{D}}, O^{\mathbb{D}'}, q) \leq q \cdot \text{SD}(\mathbb{D}, \mathbb{D}') \quad (3)$$

**Theorem 1 (Pseudorandomness of PUF-FE-composition).** Let $\mathcal{F}$ be a family of PUF-FE-compositions $E \circ \Pi$ where $\Pi$ is a $(\ell, m, \delta, q, \mu)$-PUF and $E$ is a $(m, n, \delta, \mu, \epsilon)$-fuzzy extractor. Then the advantage of distinguishing the two oracles from Definition 6 with up to $q$ queries is at most

$$\text{Adv}(O^F, O^{\text{PRF, } F}, q) \leq q \cdot \epsilon \quad (4)$$

**Proof.** By definition, each of the $q$ responses of the PUF has a min-entropy of at least $\mu$. Hence, according to the definition of fuzzy extractors, the outputs are nearly uniform even if the helper data is known. More precisely, it holds for any output $(z_i, \omega_i)$, $i = 1, \ldots, q$ that $\text{SD}((z_i, \omega_i), (\mathbb{U}_n, \omega_i)) \leq \epsilon$. Then, the claims follows from Proposition 2 and the fact that the $q$ responses are independent. \qed

**Definition 7 (PUF-PRFs).** A $(\ell, n, \delta, q, \mu, \epsilon)$-PUF-PRF is a PUF-FE-composition $(E \circ \Pi)$ where $\Pi$ is a $(\ell, m, \delta, q, \mu)$-PUF and $E$ is a $(m, n, \delta, \mu, \epsilon)$-fuzzy extractor.

The last condition guarantees that the distinguisher can make at least $q$ different queries.
4 Physically Unclonable Functions

4.1 Manufacturing variability in ICs.

During the manufacturing process, subtleties of the operating conditions as well as random variations are imprinted into an integrated circuit. This phenomenon, i.e. manufacturing variability, creates minute differences in circuit parameters, e.g. capacitances, line delays, threshold voltages etc., in chips which otherwise were manufactured to be logically identical. Manufacturing variability requires additional margins to be implemented. There are numerous references modeling manufacturing variability and proposing techniques to mitigate its effects, e.g., [21–23]. However, it is well known that in smaller technology nodes (below .18µm technology), the impact of deep sub-micron process variations becomes larger and larger and cannot be removed anymore [20]. These random variations vary from chip to chip and cause it to be physically unique. A number of PUFs have been proposed that exploit the uniqueness of an IC due to these process variations. We discuss a possible implementation below.

4.2 SRAM PUFs.

In this subsection, a practical instantiation of a Physical Unclonable Function is discussed: the SRAM PUF. We start by addressing the SRAM operation and further on show how it can be used to meet the PUF conditions from Definition 3.

Fig. 1. (a) shows the logical scheme of an SRAM cell. In its most common form, it consists of two logical inverters (labeled 1 and 2) taking each others output as input. The cell assumes two logical stable states: \((AB = 01)\) and \((AB = 10)\). (b) shows the typical two-transistor CMOS implementation of an inverter, consisting of a p-channel \((P)\) and an n-channel \((N)\) transistor.
the two cross coupled inverters. This particular construction causes the circuit to assume two distinct logically stable states, commonly denoted as ‘0’ and ‘1’. By residing in one of both states, this scheme can hence effectively store one bit of information. Two additional switches (left and right in Figure 1(a)) provide read/write access to the cell. A read operation simply inspects which state the SRAM cell is in. A write operation brings the cell into a determined state specified by the value forced on the bitline.

**Definition 8 (SRAM).** A \((\ell, m)\)-Static Random Access Memory (SRAM) is defined as a \(2^{\ell} \times m\) matrix of physical SRAM cells, each storing an element from \(\{0, 1\}\).

The powerup state under noise-free conditions of the SRAM matrix is denoted as \(M \in \{0, 1\}^{2^{\ell} \times m}\). Each row of \(M\) is uniquely labeled with an element \(i\) from \(\{0, 1\}^{\ell}\), and a specific row is denoted as \(M^{(i)}\).

Note that the actual contents of \(M\) are not defined here. The mathematical properties of the noise-free powerup state \(M\) follow from the physical properties of the SRAM implementation and are described in Propositions 3 and 4 in this Section. The following theorem states how an SRAM memory can be used to construct a PUF.

**Theorem 2.** Let \(M\) be the noise-free powerup state matrix that is associated with a specific physical SRAM realization, as defined in Definition 8. Let \(e^{(x)} \in \{0, 1\}^m\) denote a random row vector representing noise. The procedure that accepts inputs \(x \in \{0, 1\}^{\ell}\) and thereupon returns the row vector \(y = M^{(x)} + e^{(x)}\), is a realization of an \((\ell, m, \delta, q, \mu)\)-PUF as defined by Definition 3.

Note that the statistical properties of \(e^{(x)}\) are not defined here, but follow from the physical properties of the SRAM implementation and its operation conditions, as described in Proposition 5 given below.

The PUF construction that is described in Theorem 2 was originally introduced and experimentally verified in [19]. It is commonly called an SRAM PUF. In the following, we will present a step by step proof sketch for this theorem. Since PUFs concern physical constructions, the proof sketch will be based on some physical concepts and assumptions. Summarized very briefly, we have to show that \(M\) contains independent random values, and that the Hamming-weight of the random vector \(e\) is bounded.

The row vector \(y = M^{(x)} + e^{(x)}\) returned by the SRAM PUF contains the states of the SRAM cells immediately after powerup of the SRAM. The powerup behavior of the SRAM PUF is largely determined by the actual physical implementation of the SRAM cells, of which the logical operation is shown in Figure 1(a). The most common physical implementation of the inverters in the

---

10 The output of the first inverter is connected to the input of the second one, and vice versa.
SRAM cell, using two transistors in CMOS technology\(^{11}\), is shown in Figure 1(b). It is however beyond the scope of this work to explain the full transistor-level operation of the cell, which leads to the determination of the powerup state. It suffices to know that the electrical behavior of a transistor is largely determined by one single parameter. This physical parameter is a constant voltage value that is determined at manufacturing time and is called the threshold voltage \((V_{Th})\). The following proposition explains the relation between the threshold voltage of the transistors in an SRAM cell, and its powerup state. It follows instantly from applying basic electrical laws to the CMOS implementation of the SRAM cell.

**Proposition 3 (Powerup state of SRAM cell).** Under noise-free conditions \((e(x)\) contains only zeros\), the powerup state of an SRAM cell is solely determined by the difference in threshold voltage between the p-channel transistors in inverters 1 and 2, denoted as \(V_{P1}^{Th}\) and \(V_{P2}^{Th}\) respectively:

\[
\begin{align*}
\text{Noise-free powerup state} &= '0', \text{ if } V_{P1}^{Th} - V_{P2}^{Th} < 0, \\
\text{Noise-free powerup state} &= '1', \text{ if } V_{P1}^{Th} - V_{P2}^{Th} > 0.
\end{align*}
\]

From this proposition, and the fact that the threshold voltages are fixed and determined at manufacturing time, it follows that every SRAM cell has a fixed powerup state under noise-free conditions.

Ideally, the threshold voltage of identically designed transistors (such as \(P_1\) and \(P_2\)) is equal. However, since \(V_{Th}\) is a parameter that arises from a physical process, it is never entirely deterministic, but it has a stochastic term due to the uncertainties in the manufacturing process. In fact, threshold voltages are, as many other physical quantities, assumed to be normally distributed random variables\(^{12}\).

The following proposition follows directly from Proposition 3 and the fact that \(V_{P1}^{Th}\) and \(V_{P2}^{Th}\) of every SRAM cell are independently drawn samples from the same normal distribution.

**Proposition 4.** Under noise-free conditions, the fixed powerup state of a random SRAM cell is a uniformly distributed random variable. Moreover, the powerup states of distinct SRAM cells are independent.

So far, any noise during powerup has been ignored. However, at temperatures above absolute zero, there will always be some amount of thermal noise on the voltages in the SRAM cells, with the noise amplitude increasing with temperature. The noise in the SRAM cell is generally modeled as an additive Gaussian noise voltage and is denoted as \(V_{noise}\). This extends Proposition 3 to:

\(^{11}\) Complementary Metal-Oxide-Semiconductor or CMOS is the most widely used semiconductor technology used in all contemporary digital devices. It uses Metal-Oxide-Semiconductor Field-Effect Transistors or MOSFETs. We consider common six-transistor SRAM cells, two transistors per inverter and one transistor to implement an access switch.

\(^{12}\) This actually follows from the Central Limit Theorem in statistics
Proposition 5 (Noisy powerup state of SRAM cell). The noisy powerup state of an SRAM cell is determined by the difference in threshold voltage between the left and right p-channel transistors $P_1$ and $P_2$, and the noise voltage $V_{\text{noise}}$, in the specified cell at the time of powerup:

\[
\begin{align*}
\text{Noisy powerup state} &= '0', \quad \text{if } V_{P_1}^{Th} - V_{P_2}^{Th} < V_{\text{noise}}, \\
\text{Noisy powerup state} &= '1', \quad \text{if } V_{P_1}^{Th} - V_{P_2}^{Th} > V_{\text{noise}},
\end{align*}
\]

with $V_{\text{noise}}$ a random variable. $V_{\text{noise}}$ is drawn independently for every cell from the normal distribution $N(0, \sigma_{\text{noise}}^2)$ and drawn again every time the cell is powered up. $\sigma_{\text{noise}}^2$ is determined by the absolute temperature of the SRAM cell.

If it is assumed that the noise-free powerup value, as specified by Proposition 3 is the correct response, the presence of noise can induce bit errors in the SRAM PUF responses. In Theorem 2, these bit errors are represented by the random noise vector $e(x)$ that is added to the noise-free powerup values contained in $M(x)$. The probability of such a bit error occurring for a single cell is given by:

\[
p_e = Pr[\text{Noise-free powerup state} \neq \text{Noisy powerup state}]
= Pr[\text{sgn}(V_{P_1}^{Th} - V_{P_2}^{Th}) \neq \text{sgn}(V_{P_1}^{Th} - V_{P_2}^{Th} - V_{\text{noise}})]
= Pr[e_i(x) = 1, \text{ with } e_i(x) \in \{0, 1\} \text{ the } i^{th} \text{ bit of } e(x)].
\]

Because $V_{\text{noise}}$ is drawn independently for every SRAM cell, possible bit errors occur independently in distinct cells.

To complete the proof sketch, the parameters $(\ell, m, \delta, q, \mu)$ of an SRAM PUF can be determined following the theoretical description above:

- The values $\ell$ and $m$ are design parameters of the SRAM memory. Mind that the size of the memory (and hence the implementation cost) rises linearly in $m$, but exponentially in $\ell$.
- From Proposition 4 it follows that the theoretical min-entropy in one SRAM PUF response is $1^{\text{SRAM cell}}$ or $m^{\text{bit response}}$, even if the whole SRAM is queried ($q = 2^{\ell}$).
- An upper bound to the number of bit errors in an $m$-bit response, $\delta$, relates to the bit error probability $p_e$. According to the Chernoff bound, $Pr[\text{more than } \delta \text{ bit errors}]$ decreases exponentially for increasing $\delta$ if $p_e < \frac{1}{2}$.

In practice, some upper bound for $Pr[\text{more than } \delta \text{ bit errors}]$ is fixed\(^{13}\) which determines $\delta$.

In [19], these theoretical values were experimentally verified.

**Experimental results.** In [19] an SRAM PUF was constructed on an FPGA. The performed experiments indicate that the bit error probability of the response

\(^{13}\) E.g. setting $Pr[\text{more than } \delta \text{ bit errors}] < 10^{-9}$ will generally assure that more than $\delta$ bit errors will never occur in practice.
bits is bounded by $p_e \leq 4\%$ when the temperature is kept constant at $20^\circ C$, and by $p_e \leq 12\%$ at large temperature variations between $-20^\circ C$ and $80^\circ C$. The probability of more than $\delta$ bit errors occurring decreases exponentially with increasing $\delta$ according to the Chernoff bound. $\delta$ is chosen high enough such that in practice, more than $\delta$ bit errors will never be observed.

Accurately determining the min-entropy from a limited amount of PUF instances and responses is unattainable. In [42] it was shown that the mean smooth min-entropy of a stationary ergodic process is equal to the mean Shannon entropy of that process. Since the SRAM PUF responses are distributed according to such a stationary distribution (as they result from a physical phenomenon) it was estimated in [43] that its Shannon entropy equals $0.95 \text{ bit/cell}$. Because the mean smooth min entropy converges to the mean Shannon entropy, it follows that $H_{\infty}^s(M(x))$ is close to $H(M(x))$. Therefore we put, $H_{\infty}^s(M(x)) = 0.95 \cdot m$.

One big advantage of SRAM PUFs is that they are intrinsically present on many digital devices. However, for security reasons, it is suggested that the SRAM used for the SRAM PUF is dedicated solely to that task. If the SRAM powerup values can be read or written from other external applications, the assumptions concerning the unpredictability of the SRAM powerup state do not hold. Similar PUF constructions with equivalent properties have been developed using other memory elements, e.g. the Butterfly PUF [44].

5 Applications

It was already pointed out in the Introduction Section that Pseudorandom Functions are a key element in many provably secure cryptographic constructions. The efficient realization of PRFs based on physical hardness assumptions, as shown in this work, leads to useful results for the implementation of PRF-based cryptographic algorithms. A number of possible applications are mentioned here:\textsuperscript{14}:

- In their revolutionary paper [35], Luby and Rackoff presented a simple method for constructing strong pseudorandom permutations from PRFs. Moreover, they showed that these strong pseudorandom permutations can be used to construct a provably secure blockcipher when enough rounds of the permutation are applied [45–48]. Using SRAM-based PUF-PRFs to construct the PRFs yields a practically feasible and provably secure blockcipher, based on the assumed hardness of predicting the SRAM powerup states.

- While a strong pseudorandom permutation can be seen as an idealized block cipher, a pseudorandom bit generator (PRBG) can be seen as an idealized keystream generator. Intuitively, a PRBG is a deterministic program used to generate bit sequences of arbitrary length which are indistinguishable from random sequences, given as input a short random sequence (the input seed). It is a long known result that PRBGs can be constructed from PRFs, and PUF-PRFs can hence also be used to construct streamciphers. Independent

\textsuperscript{14} For detailed descriptions of these applications we refer to the extended version of this paper.
of this work, in [49] a PUF-based stream cipher has been proposed. Similar to our approach, the outputs of PUFs is used to generate a keystream. However, they do not eliminate the noise completely but suggest other means like encoding the plaintext using an error correction code to make unique decryption possible.

6 Conclusions

In this work, we presented a method for constructing PRFs from physically unclonable functions. The method is based on a theoretical model for PUFs, for which we argued why certain classes are adequately modeled in this way. That is, the construction method allows for constructing real instantiations of PRFs where the security can be proven under some reasonable physical assumptions.

Of course, any physical model can only approximately describe real life. Although experiments support our model for the considered PUF implementations, more analysis is necessary. In this context it would be interesting to look for other PUF types which fit into our model or might be used for other cryptographic applications. Furthermore, a natural continuation of this works would be to explore other cryptographic schemes based of PUF-PRFs, e.g., hash functions or public key encryption.

References